QingKeV2 Microprocessor Debug Manual

V1.0

Overview

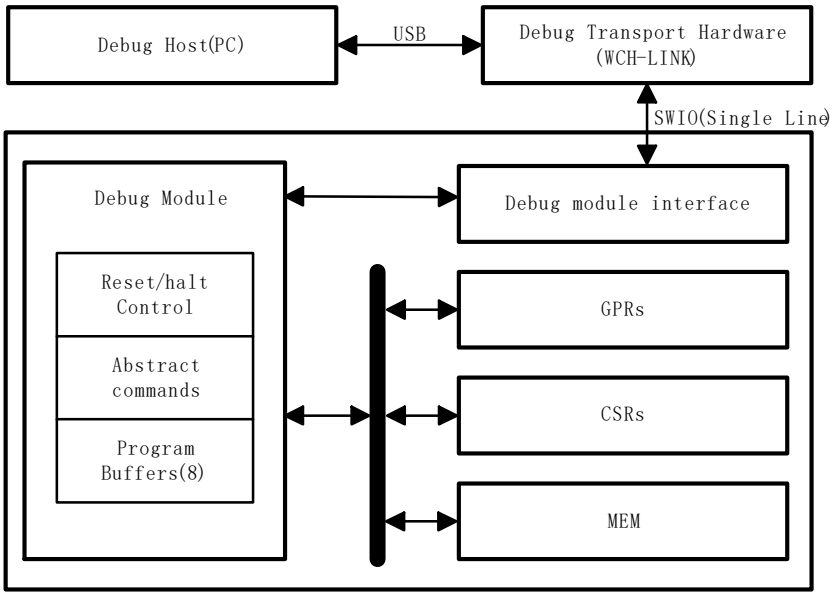
QingKe V2 series microprocessor is a 32-bit general-purpose MCU microprocessor based on the standard RISC-V instruction set RV32I subset RV32E, with only 16 general-purpose registers, half of RV32I, and a more streamlined structure for deep embedded scenarios. V2 series supports standard RV32EC instruction extensions, in addition to custom XW extensions, Hardware Prologue/Epilogue (HPE), Vector Table Free (VTF), a more streamlined single-wire serial debug interface (SDI), and support for "WFE" instructions.

QingKe V2 series microprocessor supports online debugging. The debug module conforms to the RISC-V debug specification and enables online debugging of the microprocessor through a more streamlined single-wire debug interface. This manual will introduce in detail the debug transport protocol of the debug interface, the debug module and its operation method.

Chapter 1 Overview

The simple block diagram of the debug system is shown in Figure 1-1 below. A debug module is designed inside the QingKe V2 microprocessor, and the debug module can implement functions such as halt, reset and resume of the microprocessor. It also accesses the General-purpose Registers (GPRs) and Control Status Registers (CSRs) inside the processor and memory or peripherals mapped to specific functions, etc. by means of abstract commands or Program Buffer.

Figure 1-1 Debug system block diagram



The debug module communicates with the debug transmission device through the single-wire debug interface. The debug transmission device is usually WCH-LINK, which uses USB communication with the debug host and communicates with the debug module through the single-wire interface to control and query the microprocessor status to achieve debug functions.

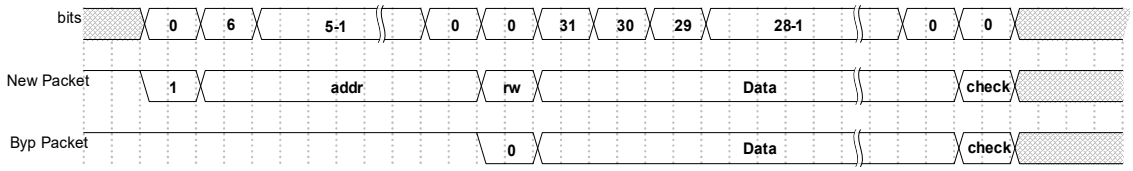
Chapter 2 Debug Transport Protocol

The debug transmission device and the debug module communicate with each other using single-line transmission. The transmission protocol defines the packet format for accessing the relevant registers of the debug module, which is described in detail as follows.

2.1 Packet Format

The debug module register operations need to be accessed in the following format, mainly including two packet types New Packet and Bypass Packet.

Figure 2-1 Data packet format



New Packet composition:

(1) 1bit start bit, fixed as data 1.
(2) 7bit address bit, set the access register address, MSB priority.
(3) 1bit read/write control bit, 1 host write, 0 host read.
(4) 32bit data, MSB priority.
(5) 1bit even parity bit, this bit is optional data bit, if the stop character is sent directly after the last bit of the data bit, then no parity bit is transmitted.

Bypass Packet composition:

(1) 1bit start bit, fixed as data 0.
(2) 32bit data bits, MSB priority, read/write bits and register addresses are the same as the most recent New Packet transfer.
(3) 1bit even parity bit, this bit is optional data bit, if the stop character is sent directly after the last bit of the data bit, then no parity bit is transmitted.

2.2 Bit Definition

Due to the single-wire debug interface, there is only one data line, and the data bit, stop bit and reset signal are judged according to the level and duration. The bus is high when it is idle, and the reset signal can be generated when the bus low level lasts for a certain time.

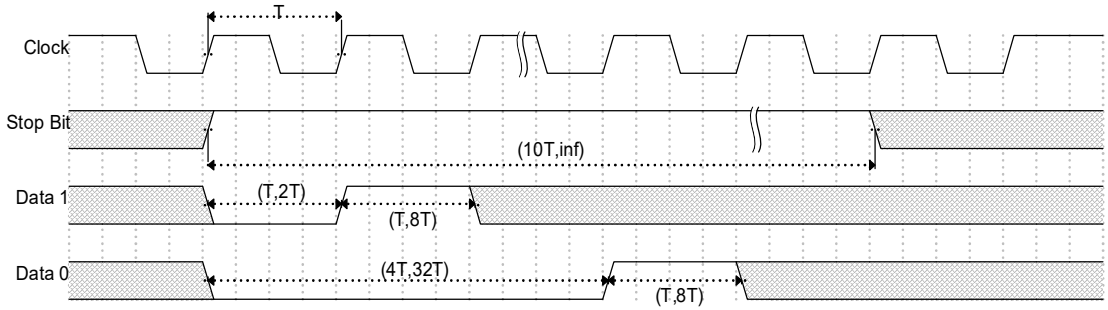
The debugging interface supports two different speed modes with slightly different timings in different modes, as detailed below.

(1) Fast mode 1x

Let the clock period of the slave debug interface be T. The timing sequence in fast mode is shown in Figure 2-2 below.

- Stop bit: a sustained high level of 10T will generate a stop bit.
- Data 1: Low level time (T, 2T), high level time (T, 8T).
- Data 0: Low level time (4T, 32T), high level time (T, 8T).

Figure 2-2 Fast mode signal timing

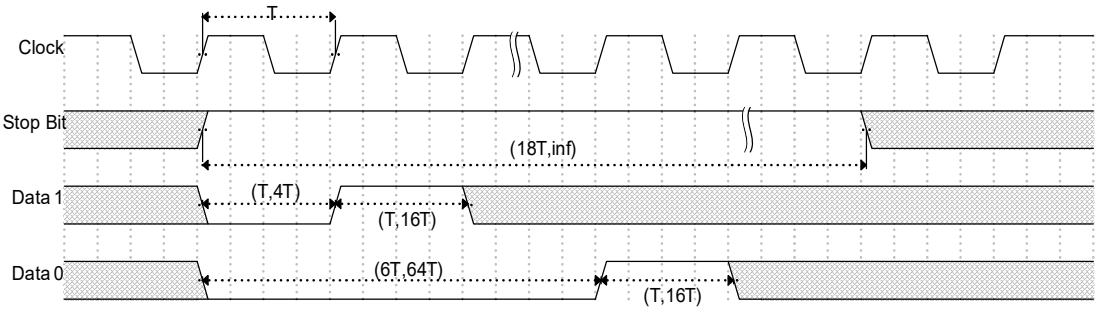


(2) Normal mode 2x

Let the clock period of the slave debug interface be T. The timing in fast mode is shown in Figure 2-3 below, which is the default mode after reset.

- Stop bit: a sustained high level of 18T will generate a stop bit.
- Data 1: Low level time (T, 4T), high level time (T, 16T).
- Data 0: Low level time (6T, 64T), high level time (T, 16T)

Figure 2-3 Medium speed mode signal timing



2.3 Debug Interface Registers

The debug interface speed mode and enable are configured by the relevant registers, which are coded using 7-bit addresses as follows.

Table 2-1 Debug interface registers

Name	Access address	Description
CPBR	0x7C	Capability Register
CFGR	0x7D	Configuration Register
SHDWCFGR	0x7E	Shadow Configuration Register

The debug interface speed mode and enable are configured by the relevant registers, which are coded using 7-bit addresses as follows.

Capability Register (CPBR)

Bit	Name	Access	Description	Reset value
[31:16]	VERSION	RO	Version number.	0x0001
[15:13]	Reserved	RO	Reserved	0
[12:11]	IOMODE	R0	Debug I/O port support mode: 00: IO_FREE mode； 01: IO_FREE and Single_IO mode； 10: IO_FREE, Single_IO, Dual_IO mode 11: IO_FREE, Single_IO, Dual_IO, Quad_IO mode。 *Note: The current version only supports IO_FREE mode.*	0
10	OUTSTA	RO	Output function status. 0: The debug slave does not have output function. 1: The debug slave has output function.	0
9	CMDEXTENSTA	R0	Command code extension function. 0: The debug slave does not have command code extension function. 1: The debug slave has command code extension function. *Note: This feature is not supported in the current version.*	0
8	CHECKSTA	RO	CRC8 checksum function. 0: The debug slave does not have CRC8 checksum function, only even checksum. 1: The debug slave supports CRC8 and even parity. *Note: The current version only supports even parity.*	0
[7:6]	Reserved	RO	Reserved	
[5:4]	SOPN	RO	Stop sign factor. 00: At least 8 times the time base. 01: At least 16 times the time base. 10: At least 32 times the time base. 11: At least 64 times the time base. *Note: Currently, only 8x mode is supported.*	0
[3:2]	Reserved	RO	Reserved	0
[1:0]	TDIV	RO	Clock division factor: 00: Divided by 1; 01: Divided by 2; Others: Reserved.	0b01

Configuration Register (CFGR)

Bit	Name	Access	Description	Reset value
[31:16]	KEY	RW	Key register: Write the register to 0x5AA5 simultaneously.	0
[15:13]	Reserved	RO	Reserved	0
[12:11]	IOMODECFG	RW	Debug I/O port support mode: 00: IO_FREE mode;	0

			01: IO_FREE and Single_IO mode; 10: IO_FREE, Single_IO, Dual_IO mode; 11: IO_FREE, Single_IO, Dual_IO, Quad_IO mode. *Note: The current version only supports IO_FREE mode.*	
10	OUTEN	RW	Output function status. 0: The debug slave does not have output function. 1: The debug slave has output function.	0
9	CMDEXTEN	RW	Command code extension function. 0: The debug slave does not have command code extension function. 1: The debug slave has command code extension function. *Note: This feature is not supported in the current version.*	0
8	CHECKEN	RW	CRC8 checksum function. 0: The debug slave does not have CRC8 checksum function, only even checksum. 1: The debug slave supports CRC8 and even parity. *Note: The current version only supports even parity.*	0
[7:6]	Reserved	RO	Reserved	
[5:4]	SOPNCFG	RO	Stop sign factor. 00: At least 8 times the time base. 01: At least 16 times the time base. 10: At least 32 times the time base. 11: At least 64 times the time base. *Note: Currently, only 8x mode is supported.*	0
[3:2]	Reserved	RO	Reserved	0
[1:0]	TDIVCFG	RO	Clock division factor: 00: Divided by 1; 01: Divided by 2; Others: Reserved.	0

Shadow Configuration Register (SHDWCFGR)

Bit	Name	Access	Description	Reset value
[31:16]	KEY	RW	Key register: Write the register to 0x5AA5 simultaneously.	0
[15:13]	Reserved	RO	Reserved	0
[12:11]	IOMODECFG	RW	Debug I/O port support mode: 00: IO_FREE mode; 01: IO_FREE and Single_IO mode; 10: IO_FREE, Single_IO, Dual_IO mode 11: IO_FREE, Single_IO, Dual_IO, Quad_IO mode。 *Note: The current version only supports IO_FREE mode.*	0
10	OUTEN	RW	Output function status. 0: The debug slave does not have output function. 1: The debug slave has output function.	0

9	CMDEXTEN	RW	Command code extension function. 0: The debug slave does not have command code extension function. 1: The debug slave has command code extension function. *Note: This feature is not supported in the current version.*	0
8	CHECKEN	RW	CRC8 checksum function. 0: The debug slave does not have CRC8 checksum function, only even checksum. 1: The debug slave supports CRC8 and even parity. *Note: The current version only supports even parity.*	0
[7:6]	Reserved	RO	Reserved	
[5:4]	SOPNCFG	RO	Stop sign factor. 00: At least 8 times the time base. 01: At least 16 times the time base. 10: At least 32 times the time base. 11: At least 64 times the time base. *Note: Currently, only 8x mode is supported.*	0
[3:2]	Reserved	RO	Reserved	0
[1:0]	TDIVCFG	RO	Clock division factor: 00: Divided by 1; 01: Divided by 2; Others: Reserved.	0

2.4 Configuration Examples

The host can set the capability of the debug interface through the configuration register CFGR and the shadow configuration register SHDWCFGR in cooperation, and query whether it takes effect through the capability register CPBR. When setting, first set the corresponding bit of SHDWCFGR and then set the corresponding bit field of CFGR to set the corresponding configuration bit of the shadow configuration register to take effect, while other configuration bits remain unchanged. For example:

(1) Enable slave output

1. Set SHDWCFGR to 0x5AA50400 and enable the shadow configuration register output to position 1.
2. Set CFGR to 0x5AA50400, update the shadow configuration register output enable bit to the configuration register, and leave the other bits of the configuration register unchanged.

(2) Configure the time base crossover factor to 1 division

1. set SHDWCFGR to 0x5AA50000 and set shadow configuration register TDIVCFG to 0b00.
2. Set CFGR to 0x5AA50003, update the shadow configuration register TDIVCFG bit field to the configuration register, and leave the other bits in the configuration register unchanged.

(3) Reset debug interface

In IO_FREE mode, reset timing for the bus pull down more than 32 times the time base, and regardless of the mode, the bus pull down more than 256 times the time base, fixed can reset the debug interface. After reset, the speed is changed to two-division mode, i.e. normal mode 2x.

The debug interface clock may vary from hardware platform to hardware platform, for example, CH32V003 defaults to 3 divisions of the internal high-speed clock 24MHz for its clock, which is 8MHz.

Chapter 3 Debug Module

QingKe V2 series microprocessors include a hardware debug module that supports complex debugging operations. When the microprocessor is halted, the debug module can access the microprocessor's GPRs, CSRs, Memory, external devices, etc. through abstract commands, program buffer deployment instructions, etc. The debug module can halt and resume the microprocessor's operation.

The debug module follows the RISC-V External Debug Support Version0.13.2 specification, detailed documentation can be downloaded from RISC-V International website.

3.1 Debug Module

The debug module inside the microprocessor, capable of performing debug operations issued by the debug host, includes.

- Access to registers through the debug interface
- Reset, halt and resume the microprocessor through the debug interface
- Read and write memory, instruction registers and external devices through the debug interface
- Deploy multiple arbitrary instructions through the debug interface
- Set software breakpoints through the debug interface
- Support abstract command auto-execution
- Support single-step debugging

The internal registers of the debug module use a 7-bit address code, and the following registers are implemented inside QingKe V2 series microprocessors.

Table 3-1 Debug module register list

Name	Access address	Description
data0	0x04	Data register 0, can be used for temporary storage of data
data1	0x05	Data register 1, can be used for temporary storage of data
dmcontrol	0x10	Debug module control register
dmstatus	0x11	Debug module status register
hartinfo	0x12	Microprocessor status register
abstractcs	0x16	Abstract command status register
command	0x17	Abstract command register
abstractauto	0x18	Abstract command auto-execution
progbuf0-7	0x20-0x27	Instruction cache registers 0-7
haltsum0	0x40	Halt status register

The debug host can control the microprocessor's halt, resume, reset, etc. by configuring the dmcontrol register. The RISC-V standard defines three types of abstract commands: access registers, fast access, and access memory. QingKe V2 microprocessor supports register (GPRs, CSRs, FPRs) access through abstract commands.

The debug module implements eight instruction cache registers progbuf0-7, and the debug host can cache multiple instructions (which can be compressed instructions) to the buffer, and can choose to continue to execute the instructions in the instruction cache registers after executing the abstract command or execute the cached instructions directly. It should be noted that if the instruction in progbufs is less than 32 bytes, the last instruction needs to be an "ebreak" or "c.ebreak" instruction, and if the instruction fills 32 bytes, the debug module

automatically adds an "ebreak" instruction. The debug host can access the abstract command and the instructions cached in the progbufs, and also the storage, peripherals, etc.

Each register is described in detail as follows.

Abstract Data 0 (data0)

Table 3-2 data0 definition

Bit	Name	Access	Description	Reset Value
[31:0]	data0	RW	Data register 0, used for temporary storage of data	0

Abstract Data 1 (data1)

Table 3-3 data1 definition

Bit	Name	Access	Description	Reset Value
[31:0]	data1	RW	Data register 1, used for temporary storage of data	0

Debug Module Control (dmcontrol)

This register controls the halt, reset, and resume of the microprocessor. Debug host write data to the corresponding field to achieve halt (haltreq), reset (ndmreset), resume (resumereq). You describe into the following.

Table 3-4 dmcontrol definition

Bit	Name	Access	Description	Reset Value
31	haltreq	WO	0: Clear the halt request 1: Send a halt request	0
30	resumereq	W1	0: Invalid 1: Restore the current microprocessor *Note: Write 1 is valid and the hardware is cleared after the microprocessor is recovered*	0
29	Reserved	RO	Reserved	0
28	ackhavereset	W1	0: Invalid 1: Clear the haverest status bit of the microprocessor	0
[27:2]	Reserved	RO	Reserved	0
1	ndmreset	RW	0: Clear reset 1: Reset the entire system other than the debug module	0
0	dmactive	RW	0: Reset debug module 1: Debug module works properly	0

Debug Module Status (dmstatus)

This register is used to indicate the status of the debug module and is a read-only register with the following description of each bit.

Table 3-5 dmstatus definition

Bit	Name	Access	Description	Reset Value
[31:20]	Reserved	RO	Reserved	0
19	allhavereset	RO	0: Invalid 1: Microprocessor reset	0

18	anyhavereset	RO	0: Invalid 1: Microprocessor reset	0
17	allresumeack	RO	0: Invalid 1: Microprocessor reset	0
16	anyresumeack	RO	0: Invalid 1: Microprocessor reset	0
[15:14]	Reserved	RO	Reserved	0
13	allavail	RO	0: Invalid 1: Microprocessor is not available	0
12	anyavail	RO	0: Invalid 1: Microprocessor is not available	0
11	allrunning	RO	0: Invalid 1: Microprocessor is running	0
10	anyrunning	RO	0: Invalid 1: Microprocessor is running	0
9	allhalted	RO	0: Invalid 1: Microprocessor is in suspension	0
8	anyhalted	RO	0: Invalid 1: Microprocessor out of suspension	0
7	authenticated	RO	0: Authentication is required before using the debug module 1: The debug module has been certified	0x1
[6:4]	Reserved	RO	Reserved	0
[3:0]	version	RO	Debug system support architecture version 0010: V0.13	0x2

Hart Info (hartinfo)

This register is used to provide information about the microprocessor to the debug host and is a read-only register with each bit described as follows.

Table 3-6 hartinfo definition

Bit	Name	Access	Description	Reset Value
[31:24]	Reserved	RO	Reserved	0
[23:20]	nscratch	RO	Number of dscratch registers supported	0x2
[19:17]	Reserved	RO	Reserved	0
16	dataaccess	RO	0: Data register is mapped to CSR address 1: Data register is mapped to memory address	0x1
[15:12]	datasize	RO	Number of data registers	0x2
[11:0]	dataaddr	RO	Data register data0 offset address, the base address is 0xe0000000	0x0f4

Abstract Control and Status (abstractcs)

This register is used to indicate the execution of the abstract command. The debug host can read this register to know whether the last abstract command is executed or not, and can check whether an error is generated during the execution of the abstract command and the type of the error, which is described in detail as follows.

Table 3-7 abstractcs definitions

Bit	Name	Access	Description	Reset Value
[31:29]	Reserved	RO	Reserved	0
[28:24]	progbufsize	RO	Indicates the number of program buffer program cache registers	0x8
[23:13]	Reserved	RO	Reserved	0
12	busy	RO	0: No abstract command is executing 1: There are abstract commands being executed *Note: After execution, the hardware is cleared.*	0
11	Reserved	RO	Reserved	0
[10:8]	cmder	RW	Abstract command error type 000: No error 001: Abstract command execution to write to command, abstractcs, abstractauto registers or read and write to data and progbuf registers 010: Does not support current abstract command 011: Execution of abstract command with exception 100: The microprocessor is not halted or unavailable and cannot execute abstract commands 101: Bus error 110: Parity bit error during communication 111: Other errors *Note: For bit writing 1 is used to clear the zero.*	0
[7:4]	Reserved	RO	Reserved	0
[3:0]	datacount	RO	Number of data registers	0x2

Abstract Command (command)

The debug host can access the GPRs, FPRs, and CSRs registers inside the microprocessor by writing different configuration values in the abstract command registers.

When accessing the registers, the command register bits are defined as follows.

Table 3-8 Definition of command when accessing registers

Bit	Name	Access	Description	Reset Value
[31:24]	cmdtype	WO	Abstract command type 0: Access register 1: Quick access (not supported) 2: Access to memory (not supported)	0
23	Reserved	WO	Reserved	0
[22:20]	aarsize	WO	Access register data bit width 000: 8-bit	0

			001: 16-bit 010: 32-bit 011: 64-bit (not supported) 100: 128-bit (not supported) *Note: When accessing floating-point registers FPRs, only 32-bit access is supported.*	
19	aarpostincrement	WO	0: No effect 1: Automatically increase the value of regno after accessing the register	0
18	postexec	WO	0: No effect 1：Execute the abstract command and then execute the command in progbuf	0
17	transfer	WO	0: Do not execute the operation specified by write 1: Execute the manipulation specified by write	0
16	write	WO	0: Copy data from the specified register to data0 1: Copy data from data0 register to the specified register	0
[15:0]	regno	WO	Specify access registers 0x0000-0x0fff are CSRs 0x1000-0x101f are GPRs 0x1020-0x103f are FPRs (not supported by V2)	0

Abstract Command Autoexec (abstractauto)

This register is used to configure the debug module so that the abstract command can be executed again when reading and writing to the progbufx and datax of the debug module, which is described as follows.

Table 3-9 abstractauto definition

Bit	Name	Access	Description	Reset Value
[31:16]	autoexecprogbuf	RW	If a position 1, the corresponding read or write to progbufx register will cause the abstract command in the command register to be executed again. *Note: V2 series design 8 progbuf, corresponding to bits [23:16].*	0
[15:12]	Reserved	RO	Reserved	0
[11:0]	autoexecdata	RW	If a position 1, the corresponding read or write to datax register will cause the abstract command in the command register to be executed again. *Note: V2 series design 2 data registers, corresponding to bits [1:0].*	0

Program Buffer X (progbufx)

This register is used to store any instruction, deploy the corresponding operation, including 8, need to pay attention to the last execution needs to be "ebreak" or "c.ebreak".

Table 3-10 progbuf definition

Bit	Name	Access	Description	Reset Value
[31:0]	progbuf	RW	Instruction encoding for cache operations, which may include compression instructions	0

Halt Summary 0 (haltsum0)

This register is used to indicate whether the microprocessor is halted or not. Each bit indicates the halted status of a microprocessor, and when there is only one core, only the lowest bit of this register is used to indicate it.

Table 3-11 haltsum0 definition

Bit	Name	Access	Description	Reset Value
[31:1]	Reserved	RO	Reserved	0
0	haltsum0	RO	0: Microprocessor operates normally 1: Microprocessor stop	0

In addition to the above-mentioned registers of the debug module, the debug function also involves some CSR registers, mainly the debug control and status register dcsr and the debug instruction pointer dpc, which are described in detail as follows.

Debug Control and Status (dcsr)

Table 3-12 dcsr definition

Bit	Name	Access	Description	Reset Value
[31:28]	xdebugver	DRO	0000: External debugging is not supported 0100: Support standard external debugging 1111: External debugging is supported, but does not meet the specification	0x4
[27:16]	Reserved	DRO	Reserved	
15	ebreakm	DRW	0: The ebreak command in machine mode behaves as described in the privilege file 1: The ebreak command in machine mode can enter debug mode	0
[14:13]	Reserved	DRO	Reserved	0
12	ebreaku	DRW	0: The ebreak command in user mode behaves as described in the privilege file 1: The ebreak command in user mode can enter debug mode	0
11	stepie	DRW	0: Interrupts are disabled under single-step debugging 1: Enable interrupts under single-step debugging	0
10	Reserved	DRO	Reserved	0
9	stoptime	DRW	0: System timer running in Debug mode 1: System timer stop in Debug mode	0
[8:6]	cause	DRO	Reasons for entering debugging 001: Entering debugging in the form of ebreak command (priority 3) 010: Entering debugging in the form of trigger module	0

			(priority 4, the highest) 011: Entering debugging in the form of halt request (priority 1) 100: Entering debugging in the form of single-step debugging (priority 0, the lowest) 101: Entering debug mode directly after microprocessor reset (priority 2) Others: Reserved.	
[5:3]	Reserved	DRO	Reserved	0
2	step	DRW	0: Turn off single-step debugging 1: Enable single-step debugging	0
[1:0]	prv	DRW	Privilege mode 00: User mode 01: Supervisor mode (not supported) 10: Reserved 11: Machine mode *Note: Record the privileged mode when entering debug mode, the debugger can modify this value to modify the privileged mode when exiting debug.*	0

Debug PC (dpc)

This register is used to store the address of the next instruction to be executed after the microprocessor enters debug mode, and its value is updated with different rules depending on the reason for entering debug. dpc register is described in detail as follows.

Table 3-13 dpc definitions

Bit	Name	Access	Description	Reset Value
[31:0]	dpc	DRW	Command Address	0

The rules for updating the registers are shown in the following table.

Table 3-14 dpc update rules

Enter the debug method	dpc Update rules
ebreak	Address of the Ebreak instruction
single step	Instruction address of the next instruction of the current instruction
trigger module	Temporarily not supported
halt request	Address of the next instruction to be executed when entering Debug

3.2 Operation Examples

Halt Microprocessor

This process is used to halt the microprocessor and proceeds as follows.

Table 3-15 Microprocessor suspension process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.

Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmstatus(0x11)	R-0	rdata	Get the debug module status information, check rdata[9:8], if the value is 0b11, it means the processor enters the halt state normally, otherwise it means it does not enter the halt state and needs to continue to send the halt request to check the status.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request bit.

Resume Microprocessor

This process is used to resume a microprocessor that is in suspension, and the steps are as follows.

Table 3-16 Microprocessor resume process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request bit.
Dmcontrol(0x10)	W-1	0x40000001	Initiate a resume request. The resume request bit is valid for writing 1, and the hardware clears 0 after recovery.
Dmstatus(0x11)	R-0	rdata	Get the debug module status information, check rdata[17:16], if the value is 0b11, it means the processor has recovered.

Reset Microprocessor

This process is used to reset the microprocessor, after which the microprocessor can either enter halt mode again or start running again, as follows.

(1) Microprocessor re-runs after reset

Table 3-17 Microprocessor reset and run process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request bit.
Dmcontrol(0x10)	W-1	0x00000003	Initiate a core reset request.
Dmstatus(0x11)	R-0	rdata	Get the debug module status information, check rdata[19:18], if the value is 0b11, it means the processor has been reset, otherwise the reset failed.
Dmcontrol(0x10)	W-1	0x00000001	Clear the reset signal.
Dmcontrol(0x10)	W-1	0x10000001	Clear the reset status signal, this bit is valid for write 1 and read constant 0.
Dmstatus(0x11)	R-0	rdata	Get the debug module status information, check rdata[19:18], if the value is 0b00, it means the processor reset status has been cleared, otherwise the clearing fails.

(2) Microprocessor halted immediately after reset

Table 3-18 Continued halt process after microprocessor reset

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x80000003	Initiate a core reset request and hold the halt request.
Dmstatus(0x11)	R-0	rdata	Get the debug module status information, check rdata[19:18], if the value is 0b11, it means the processor has been reset, otherwise the reset failed.
Dmcontrol(0x10)	W-1	0x80000001	Clear the reset signal and hold the halt request.
Dmcontrol(0x10)	W-1	0x90000001	Clear the reset status signal and hold the halt request.
Dmstatus(0x11)	R-0	rdata	Get the debug module status information, check rdata[19:18], if the value is 0b00, it means the processor reset status has been cleared, otherwise the clearing fails.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request when the processor is reset and haltd again.

Reset Debug module

This process resets only the debug module, as detailed below.

Table 3-19 Debug module reset process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Dmcontrol(0x10)	W-1	0x00000003	Write command.
Dmcontrol(0x10)	R-0	rdata	Check if rdata is the value written in the previous step.
Dmcontrol(0x10)	W-1	0x00000002	Write the debug module reset command.
Dmcontrol(0x10)	R-0	rdata	Check whether rdata[1] is 0b0, if it is, the reset is successful, otherwise the reset fails.

Read/write General-purpose Registers (GPR)

The abstract command supports reading and writing to the general-purpose registers of the microprocessor, and the detailed process is as follows.

(1) Read GPR, take x6 as an example

Table 3-20 Read GPR process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Data0(0x04)	W-1	0x00000000	Clear the Data0 register.

Command(0x17)	W-1	CMD (0x00221006)	Set abstract command to copy x6 data to Data0 register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Data0(0x04)	R-0	rdata	Read the Data0 data rdata, which is the x6 register value.

(2) Write GPR, take x6 as an example

Table 3-21 Writing GPR process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Data0(0x04)	W-1	wdata	Write the data to be written, wdata, to Data0.
Command(0x17)	W-1	CMD (0x00231006)	Set abstract command to copy Data0 data to x6 register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.

Read/write Control and Status Registers (CSR)

The abstract command also supports reading and writing to the microprocessor's control and status registers, with the following detailed process.

(1) Read CSR, take mepc for example, its CSR address is 0x341

Table 3-22 Read CSR process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Data0(0x04)	W-1	0x00000000	Clear the Data0 register.
Command(0x17)	W-1	CMD (0x00220341)	Set the abstract command to copy the data in CSR 0x341 to Data0 register.

Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Data0(0x04)	R-0	rdata	Read the Data0 data rdata, which is the CSR 0x341 register value.

(2) Write CSR, take mepc as an example, its CSR address is 0x341

Table 3-23 Writing CSR process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Data0(0x04)	W-1	wdata	Write the data to be written, wdata, to Data0.
Command(0x17)	W-1	CMD (0x00230341)	Set the abstract command to copy Data0 data to CSR 0x341 register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.

Read/write Memory

V2 microprocessor debug module abstract command only supports the register access mode, solid for a memory address to read and write the way to use the abstract command and pre-set progbufx instructions, read and write. Note that when there are 8 progbufx, which can store a total of 32B bytes of instructions, when less than 32B, the last instruction is required to be an "ebreak" instruction, and when 32B is stored, the module automatically adds the "ebreak" instruction at the end. "instruction, when the specific following.

(1) Memory reading

For FLASH, RAM, MCU peripheral registers, etc., all can be read using memory read mode, taking address 0x20000000 as an example.

Table 3-24 Read memory process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Progbuf0(0x20)	W-1	wcode	Write the machine code wcode of the instruction to be executed to progbuf0, for example, "lw x6,0(x5)" wcode is

			0x0002a303.
Progbuf1(0x21)	W-1	0x00100073	If it is less than 32B, add the "ebreak" instruction to progbuf1.
Data0(0x04)	W-1	0x20000000	Write the address to be read to Data0.
Command(0x17)	W-1	CMD (0x00271005)	Set the abstract command to copy the Data0 data to the x5 register and set the abstract command to execute the instruction in progbufx after execution.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Command(0x17)	W-1	CMD (0x00221006)	Set the abstract command to copy the x6 register value to Data0.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Data0(0x04)	R-0	rdata	Read the Data0 data rdata, which is the 0x20000000 address value. It should be noted that the method operates the register value, the corresponding register value should be saved before the operation, and restored after the operation.

(2) Memory writing

For RAM, MCU peripheral registers, etc. can be written directly according to the following process. For FLASH writing, it is necessary to program its registers in accordance with the FLASH controller requirements of different chips by step.

Table 3-25 Write memory process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Progbuf0(0x20)	W-1	wcode	Write the machine code wcode of the instruction to be executed to progbuf0, e.g. "sw x7,0(x5)" wcode is 0x0072a023.
Progbuf1(0x21)	W-1	0x00100073	If it is less than 32B, add the "ebreak" instruction to progbuf1.

Data0(0x04)	W-1	wdata	Write the address to be written, to Data0, for example 0x20000000.
Command(0x17)	W-1	CMD (0x00231005)	Set abstract command to copy Data0 data to x5 register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Data0(0x04)	W-1	wdata	Write the data to be written, wdata, to the Data0 register.
Command(0x17)	W-1	CMD (0x00271007)	Set the abstract command to write Data0 data to the x7 register, and set the program in progbufx to be executed automatically after the abstract command is executed.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.

Single-step Execution

By setting the status and control registers under debug, single-step execution in debug mode can be realized and whether interrupts are enabled under single-step can be controlled, as detailed below.

Table 3-26 Single-step execution process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Data0(0x04)	W-1	0x00008007	Write the DCSR register value to be written to the Data0 register, here set dcsr.breakm=1, dcsr.stepie=0, dcsr.step=1, dcsr.prv=3 to enable the machine mode to execute ebreak into debug, single step under the interrupt is prohibited, set single step execution.
Command(0x17)	W-1	0x002307b0	Copy the data in Data0 to DCSR register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly,

			check to fix the error according to the error type.
Dmcontrol(0x10)	W-1	0x40000001	Initiate a resume request. The resume request bit is valid for writing 1, and the hardware clears 0 after recovery.
Dmstatus(0x11)	R-0	rdata	Get debug module status information.
Data0(0x04)	W-1	0x00000003	When the microprocessor enters a halt again, the DCSR configuration value that turns off single-step debugging is written to Data0.
Command(0x17)	W-1	0x002307b0	Copy the data in Data0 to DCSR register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Dmcontrol(0x10)	W-1	0x40000001	Initiate a resume request. The resume request bit is valid for writing 1, and the hardware clears 0 after recovery.

Set Software Breakpoints

When debugging the QingKe V2 microprocessor online, you can insert a breakpoint by inserting the "ebreak" instruction into the program. The reference procedure is as follows.

Table 3-27 Setting software breakpoints process

Debug register address	R/W	Value	Description
Dmcontrol(0x10)	W-1	0x80000001	Make the debug module work properly.
Dmcontrol(0x10)	W-1	0x80000001	Initiate a halt request.
Dmcontrol(0x10)	W-1	0x00000001	Clear the halt request.
Data0(0x04)	W-1	0x00008003	Write the software breakpoint DCSR configuration value to the Data0 register.
Command(0x17)	W-1	0x002307b0	Copy the data in Data0 to DCSR register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Dmcontrol(0x10)	W-1	0x40000001	Initiate a resume request, run to insert breakpoint ebreak and automatically halt to enter debug mode.
Dmstatus(0x11)	R-0	rdata	Get debug module status information until execution reaches a breakpoint and then enters halt again.
Data0(0x04)	W-1	0x00000003	When the microprocessor enters a halt again, the DCSR configuration value that turns off single-step debugging is written to Data0.

Command(0x17)	W-1	0x002307b0	Copy the data in Data0 to DCSR register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Data0(0x04)	W-1	wpc	Write the value of the next program pointer wpc that needs to be continued after the breakpoint to the Data0 register.
Command(0x17)	W-1	0x002307b1	Copy Data0 data to the pointer DPC register.
Abstracts(0x16)	R-0	rdata	Check rdata[12], i.e. whether the query busy bit is 0b1, if yes, it means the abstract command is executing, otherwise it means no abstract command is executing; check rdata[10:8], i.e. whether the cmderr value is 0b000, if yes, the abstract command is executing normally, otherwise the abstract command is executing incorrectly, check to fix the error according to the error type.
Dmcontrol(0x10)	W-1	0x40000001	A resume request is initiated and the program continues from the DPC pointer position.